AMENDMENTS TO THE SPECIFICATION:

Please amend the specification as follows:

Please replace the paragraph appearing on page 1, lines 3-6, with the following paragraph:

This application is a Continuation-in-Part ~~application~~ of U.S. application Ser. No. 09/672,562, filed Sep. 28, 2000, now abandoned, which claims priority to U.S. Provisional Application No. 60/162,825, filed Nov. 1, 1999, the entire teachings of which are incorporated herein by reference.

Please replace the paragraph appearing on page 1, lines 14-20 (as amended in the Amendment filed June 22, 2004, with the following paragraph:

~~An alternative~~ Such a data structure ~~which~~ has, for example, been supported in the [incr Tk] language, which allows values to be stored in strings or arrays as options associated with an instance object.

Please replace the paragraph appearing on page 4, line 27, through page 5, line 4, with the following paragraph:

The preferred embodiment of the invention is presented with respect to the Curl™ language developed by Curl Corporation of Cambridge, Mass. The Curl™ programming language is based upon a language developed at Massachusetts Institute of Technology and presented in "Curl: A Gentle Slope Language for the Web," WorldWideWeb Journal, by M. Hostetter et al., Vol II. Issue 2, O'Reilly & Associates, Spring 1997.

Please replace the paragraph appearing on page 16, lines 10-25, with the following paragraph:

When the compiler compiles a reference to an option (e.g., x.length), it operates as in Figure 3A, using the subroutine get-option-binding of Figure 3B to find the most specific option binding b. At 31, c is defined as the class type object that represents x's type, n is the option name, such as length, for which the most specific option binding is being searched. At 31, the subroutine get-option-binding of Figure 3B is called. This is a recursive routine which first searches through the option bindings list of the class c and then, if an option binding is not found, is called again to search base classes, that is, immediate parents of the class object. At 33, b is defined as the option bindings pointer of class c which points to the linked list of option bindings. For example, if the class type is object 40 of Figure 2A, the option binding pointer is a pointer to option binding 34. At 35, if there is an options binding list, b is not null and the name of the first option binding is compared to the name being searched at 37. If the names match, the most specific option binding has been located and is returned, at 39, to 31 of Figure 3A. If the names do not match at 37, b is changed at 41 to the next pointer value of the option binding considered. Thus, the system loops through 35, 37 and 41 until a match is found with the option name or the option binding list is completed.

Please replace the paragraph appearing on page 17, lines 9-13, with the following paragraph:

Once the get-option-binding routine is complete, either an option binding or null has been returned at 57. If null is returned, a compilation error "Class member n not found" is returned at 59. If an option binding was located, code for the requested option operation is emitted using the returned option binding as the most specific option binding at 61.

Please replace the paragraph appearing on page 18, line 27, through page 19, line 3, with the following paragraph:

For example, Figure 2F illustrates the case of an instance object [[43]] 44 which points to a class type D. Class type D inherits from a class E which, in turn, inherits from OptionHashTable. The options pointer of instance [[43]] 44 points to an Option Table [[45]] 63, rather than a linked list. Option Table [[45]] 63 includes pointers to base option bindings of class E as keys and includes specific values for those keys. Only keys for which values have been set are included in the table.

Please replace the paragraph appearing on page 27, line 25, through page 28, line 5, with the following paragraph:

In the case of option item 132, the option parent pointer leads to the VBox instance 130. If at 152 it is determined that there is no parent instance, the default value of the base option binding is returned at 154. In the above example, the value 10.0 would be returned from option binding 122. However, if as in the example of instance 132 there is an option-parent, the index p becomes the option pointer of that parent and any linked list of that parent is followed in the loop of 142, 144, 148. In the example of Figure 10, the instance 130 points to option item 136 which, at 144, is noted to have a matching key, so the value 24.0 is returned at 146.

Please replace the paragraph appearing on page 29, lines 19-28, with the following paragraph:

The operation of x.register-options when x has no option children proceeds via Figures 13A and 13B. In Figure 13A, at 200, the variable c is defined as the class type object that represents the type of x, e.g., the type of TextLabel 134 that has no children. An empty list l of option bindings is established and is to be filled through a recursive process in which the option bindings of each class ancestor of the object, such as TextLabel 134, are checked to determine whether they include nonlocal options of c with change handlers. To perform that recursive search, a subroutine {register-class-options c,1pk } of Figure 13B is called. The completed list is a list of base option bindings for which this particular object, for example, TextLabel 134, is registering with the object OptionList for notification of any changes.

Please replace the paragraph appearing on page 30, lines 14-29, with the following paragraph:

At 210, the system continues to follow the option binding list of class type 118 following the next pointers. Through this loop, each option binding of class type 118 which is nonlocal and has a change handler is added to the list. When that option binding list is complete, the system defines a set bcs which contains the base classes of class type 118, the immediate parents of class 118 from which the class directly inherits, at 212. The number of those class types is defined as the limit. In this case, class type 118 has only one parent, class type Graphic 116, which is included in the set bcs. At 214, the limit is greater than zero so the limit has not been reached. Thus, at 216 the subroutine register-class-options is called for the specified parent. As a result, all of the parents of the parent 116 are ultimately processed recursively. Specifically, at 216 the option bindings of class type 116 are checked and, through that pass of the subroutine register-class-options, all parents of class type 116 are similarly checked. Once all of the ancestors of class type 116 have been checked through subroutine calls and all of the appropriate option bindings have been added to the list l, the index i is incremented at 218. In this case, there was only one parent to class type 118, so the limit is reached at 214 and the list is returned to OptionList at 220.

Please replace the paragraph appearing on page 32, lines 12-17, with the following paragraph:

Once the list l is complete, the list is used at 248 to complete the registered options list 236 for object 230, and the registration process is done at 250. Further, the option-reregister-request is next applied to the graphical parent, in this case the graphical parent of VBox 230. Thus, the reregistration process is required through the full ancestry of the graphical hierarchy, and at each object in that ancestry, all graphical children are caused to search their class hierarchies for nonlocal options having change handlers.

Please replace the paragraph appearing on page 33, line 11, through page 34, line 1, with the following paragraph:

The subroutine {nonlocal-option-change-notify x, bb, v} runs the change handlers on the object x for the option whose base option binding is bb, to notify the object's change handlers that the value of this option has become v. The change handlers of x's option children and their descendants are also executed if appropriate. This subroutine operates as in Figure 17. At 270, the local-option-change-notify routine of Figure 6 is first called to process the change handlers of all of the superclasses of the object x. In addition, it is determined from the register-options list of the object, such as list 236 of object 230 in Figure 14, whether any children need to be notified of the change in the option. At 272, it is determined whether the base option binding of interest is present in the list 1. If not, the process is done. If so, all graphical children of the object are notified. At 274, all immediate option children of the object x, such as object 230, are included in the set xc. The variable nc is set at the number of children, and the index i is set at zero. At 276, nc is compared to i to determine whether there are additional children to be notified. If so, at 278 it is determined whether the option represented by bb is already set in the option list of the child. If so, no notification is required and the index i is incremented at 280. If the option represented by bb is not included in that child's list, the nonlocal-option-change-notify subroutine of Figure 17 is called for that child, at 282. As a result, that child will notify its own change handlers and also require that its children and their descendants notify their change handlers, if applicable.

Please replace the paragraph appearing on page 36, lines 4-8, with the following paragraph:

In Figure 18, each package is represented by an object 290 of type Package, which has a field "nonlocal-option-table" that points to a hash table 292. Hash table 292 maps ClassType objects to ~~Option Binding~~ OptionBinding objects 294 representing nonlocal options that should be treated as though they were part of the chain of OptionBinding objects pointed to by the option-bindings field of the ClassType object.